DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9712

CLUSTERING KNOWLEDGE IN TABULAR KNOWLEDGE BASES

by

J. Vanthienen

E. Dries

J. Keppens

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9712

CLUSTERING KNOWLEDGE IN TABULAR KNOWLEDGE BASES

by

J. Vanthienen

E. Dries

J. Keppens

D/1997/2376/14

Clustering Knowledge in Tabular Knowledge Bases

J. Vanthienen, E. Dries & J. Keppens
Katholieke Universiteit Leuven
Department of Applied Economic Sciences
Naamsestraat 69, B-3000 Leuven (Belgium)
E-mail: {Jan.Vanthienen, Elke.Dries}@econ.kuleuven.ac.be

Abstract

Recently, there has been a growing interest in maintenance and efficiency of large knowledge based systems. Decomposition of knowledge based systems is recognized as an important research issue in this respect.
In this paper, we discuss the decomposition of knowledge bases that consist of decision tables. Several algorithms to decompose large decision tables into smaller components are proposed.

1. Introduction

Knowledge based systems (KBSs) hold great potential for solving information problems, but slow execution speed, maintenance problems and the lack of rigorous verification & validation (V&V) techniques are major bottlenecks in the transition from the experimental to the operational phase.

It has been recognized in literature that the decomposition of KBSs has a favorable impact on maintenance, V&V and computational efficiency. In Hicks [2], it is indicated how the application of decomposition techniques can avoid particular types of maintenance anomalies in expert system rule bases. Moily and Murray [4] have developed a technique to modularize prolog knowledge bases in order to increase execution speed. In Vanthienen et al. [6] and Wendler and Ayel [7], the need for decomposition techniques is introduced from a V&V point of view.

Both in ***verification and validation***, modularization can be considered a very important concept. Most research in verification ('building the system right') concentrates on domain-independent techniques such as anomaly detection, aimed at detecting abuse or unusual use of the knowledge representation scheme used. Since verification algorithms, and extension checks in particular, face a combinatorial explosion as the size of the knowledge base increases, attempts to overcome this problem include partitioning the knowledge base. Although this approach dramatically reduces the time needed to check each individual module, the possibility of inter-modular anomalies that arise due to dependencies between (components of) different modules is nevertheless not ruled out. It can be easily understood that modularization theory can be of assistance in selecting a partitioning that minimizes the presence of inter-modular dependencies, thereby reducing the need for time-expensive inter-modular checks. Not only in verification, but also in validation ('building the right system'), modularization theory can play an important role, providing a basis for generating an ensemble of test cases with respect to a specific subtopic. In addition, visualization of each of these modules will facilitate direct examination of the knowledge by the expert.

This paper was presented at the Eighth IEEE International Conference on Tools with Artificial Intelligence (TAI96), November 16-19, 1996, Toulouse, France.

In this paper we look at the decomposition of knowledge bases that consist of decision tables. Decision tables store rules of the form *if **condition(s)** then **action(s)***. They have been successfully used in the construction, V&V and implementation of KBSs [1] [2] [5]. The success of decision tables is probably due to the availability of an intuitive and simple tabular representation, that facilitates correctness, completeness and consistency checking. Unfortunately, decision tables tend to grow very large as the number of relevant conditions increases. A solution is to decompose decision tables into smaller components that are easier to manage. In this paper, several decomposition algorithms are presented. They are all implemented and supported by the PROLOGA tool [5].

2. Basic concepts

In this section, the notion of decision table is formally defined. We assume the existence of a set **C** of conditions and a set **A** of actions.

Let cnum be the number of conditions, then $\mathbf{C} = \{C_i, i = 1 .. cnum\}$. Each condition C_i consists of a condition subject CS_i, a condition domain CD_i and a set of condition states CT_i: $\mathbf{C} = \{C_i = (CS_i, CD_i, CT_i), i = 1 .. cnum\}$. CS_i is the name of C_i. CD_i is the set of all possible values condition C_i can take. $CT_i = \{S_{ik}, k = 1 .. n_i\}$ is an ordered set of n_i condition states S_{ik} ($n_i \geq 2$). A condition state S_{ik} of a condition C_i is a logical expression concerning the elements of CD_i, that determines a subset S^*_{ik} of CD_i such that $CT^*_i = \{ S^*_{ik}, k = 1 .. n_i\}$, the set of all these subsets, constitutes a partition of CD_i. The condition space CR is defined as the Cartesian product of the condition state sets: $CR = CT_1 \times CT_2 \times ... \times CT_{cnum}$. An element of CR is called a *condition entry* or a *condition combination.*

Let anum be the number of actions, then $\mathbf{A} = \{A_j, j = 1 .. anum\}$. Each action A_j consists of an action subject AS_j and a set of action values AV_j: $\mathbf{A} = \{A_j = (AS_j, AV_j), j = 1 .. anum\}$. AS_j is the name of A_j. Each $AV_j = \{AV_{jl}, l = 1 .. m_j\}$ is a set of m_j action values. In this text, it is assumed that $\forall j \in \{1 .. anum\}$: $AV_j = \{., x, -\}$, with '.': unknown, 'x': execute and '-': do not execute. The action space AR is defined as the Cartesian product of the action value sets: $AR = AV_1 \times AV_2 \times ... \times AV_{anum}$. An element of AR is called an *action entry* or an *action configuration.*

A decision table is a *function* from the condition space to the action space, by which every condition combination $x \in CR$ is mapped into one (completeness) and only one (exclusivity) action configuration $z \in AR$:

$$DT: CR \rightarrow AR: x \mapsto z = DT(x)$$

The term decision table refers to the tabular denotation that is used to represent a decision table. In such a representation, the condition combinations and their corresponding action configurations are arranged in columns in lexicographical order (the condition states of the lowest condition varying first). Figure 1 shows (the tabular representation of) an example decision table. Condition and action subjects are shown in the left part of the table, while each column represents a rule of the form *if **conditions** then **actions***.

Decision tables can occur in expanded or in contracted form. In the expanded decision table, all condition combinations are enumerated, while in the contracted decision table, adjacent (groups of) columns that only differ in the state value for one condition and that contain the same action configuration are joined, thereby minimizing the number of columns. If all states of a condition can be joined, the condition entry '-' (irrelevant) is used. Figure 2 shows the decision table of figure 1 in contracted form.

1. Credit Limit	OK												Not OK											
2. ^Customer	Good						Not Good						Good						Not Good					
3. Quantity Ordered	<50		50-<150		>=150		<50		50-<150		>=150		<50		50-<150		>=150		<50		50-<150		>=150	
4. Stock Sufficient	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
1. ^Execute	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
2. Refuse	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x
3. Put on Waiting List	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-

Figure 1: Example decision table

1. Credit Limit	OK	Not OK			
2. ^Customer	-	Good			Not Good
3. Quantity Ordered	-	<50	50-<150	>=150	-
4. Stock Sufficient	-	-	Y	N	-	-
1. ^Execute	x	x	x	-	-	-
2. Refuse	-	-	-	-	-	x
3. Put on Waiting List	-	-	-	x	x	-

Figure 2: Contracted decision table

3. Structures of decision tables

Two types of structures exist: hierarchical structures and flat structures. Hierarchical structures are used to express that a decision table refines a condition or an action of another decision table, while flat structures are used to express whether decision tables have conditions and/or actions in common.

3.1. Hierarchical structures

Two types of hierarchical structures can be distinguished: the condition-head table structure and the action-head table structure.

Condition-head table structure. A condition-head table structure connects two decision tables that are called the condition table and the head table. The connection is a relation between the actions of the condition table and one of the conditions in the head table. The latter condition is called the *connecting condition*. Its subject is preceded by the symbol ^.

Let CS be the condition subject of the connecting condition with n condition states S_1, S_2, ..., S_n. Then the condition table has n actions, with action subjects: CS := S_1, CS := S_2, ..., CS := S_n. The action configurations of the condition table are such that exactly one action has value 'x' (execute), while all other actions have value '-' (do not execute). The condition table thus determines a unique value for the connecting condition, that serves as an input value for the head table. An example is shown in figure 3.

Action-head table structure. An action-head table structure connects two decision tables named action table and head table. The connection is a relation between the action table and one of the actions in the head table. The latter action is called the *connecting action*. Its subject is preceded by the symbol ^.

The action subtable is a further specification of the connecting action. It is triggered whenever the connecting action has value 'x' (execute) in the head table. Figure 4 shows an example of an action-head table structure.

CONDITION TABLE

1. Age of Account	< 1 Year		>= 1 Year
2. Turnover	<100	>=100	-
1. Customer := Good	-	x	x
2. Customer := Not Good	x	-	-

HEAD TABLE

1. Credit Limit	OK	Not OK			
2. ^Customer	-	Good			Not Good
3. Quantity Ordered	-	<50	50-<150	>=150	-
4. Stock Sufficient	-	-	Y	N	-	-
1. ^Execute	x	x	x	-	-	-
2. Refuse	-	-	-	-	-	x
3. Put on Waiting List	-	-	-	x	x	-

Figure 3: Condition-head table structure

HEAD TABLE

1. Credit Limit	OK	Not OK			
2. ^Customer	-	Good			Not Good
3. Quantity Ordered	-	<50	50-<150	>=150	-
4. Stock Sufficient	-	-	Y	N	-	-
1. ^Execute	x	x	x	-	-	-
2. Refuse	-	-	-	-	-	x
3. Put on Waiting List	-	-	-	x	x	-

ACTION TABLE

1. Travel Distance	<50	50-<100	>=100
1. Free Delivery	x	-	-
2. Railway Transport	-	-	x
3. Road Transport	x	x	-

Figure 4: Action-head table structure

3.2. Flat structures

Flat structures are used to express the relation between two decision tables that are not hierarchically related to each other. Four relations are possible.

Two decision tables are *C-connected* if they have at least one condition in common and they have no actions in common (figure 5). Two decision tables are *A-connected* if they have at least one action in common and they have no conditions in common. Two decision tables are *CA-connected* if they have at least one condition and at least one action in common. Two decision tables are *CA-disjoint* if they have neither conditions nor actions in common.

1. Credit Limit	OK	Not OK			
2. ^Customer	-	Good			Not Good
3. Quantity Ordered	-	<50	50-<150	>=150	-
4. Stock Sufficient	-	-	Y	N	-	-
1. ^Execute	x	x	x	-	-	-
2. Put on Waiting List	-	-	-	x	x	-

C

1. Credit Limit	OK	Not OK	
2. ^Customer	-	Good	Not Good
1. Refuse	-	-	x

Figure 5: Two C-connected decision tables

4. Algorithms for the decomposition of decision tables

In this section, a number of algorithms are presented that decompose a given decision table in two or more decision tables, that can be connected by means of a hierarchical structure or by flat structures. For the sake of conciseness, the conditions and actions in the decision tables that are used to illustrate the algorithms, are represented by a number, while condition states are represented by a letter.

4.1. Dependency matrix

At the basis of the algorithms is the dependency matrix of the decision table to decompose, representing which actions depend on which conditions.

Definitions. In order to define dependency of actions on conditions, projection functions are used. Assume a finite number of arbitrary sets Q_1, Q_2, ..., Q_n, $n \geq 1$. For each non-empty subset $\{i_1, i_2, ..., i_k\}$ of $\{1, 2, ..., n\}$, the projection function $\Pi_{i_1, i_2, ..., i_k}$ is defined as follows:

$$\Pi_{i_1, i_2, ..., i_k} : Q_1 \times Q_2 \times ... \times Q_n \rightarrow Q_{i_1} \times Q_{i_2} \times ... \times Q_{i_k} : (q_1, q_2, ..., q_n) \mapsto (q_{i_1}, q_{i_2}, ..., q_{i_k})$$

Assume a decision table DT with condition set **C** and action set **A**. An action A_j is *dependent* on a condition C_i iff $\exists$ (x_1, z_1), $(x_2, z_2) \in$ DT: $(\Pi_{1, ..., i-1, i+1, ..., cnum}(x_1) = \Pi_{1, ..., i-1, i+1, ..., cnum}(x_2))$ and $(\Pi_j(z_1) \neq \Pi_j(z_2))$. The *dependency matrix* D of DT is a cnum $\times$ anum matrix such that $d_{ij} = 1$ if action A_j depends on condition C_i and $d_{ij} = 0$ if action A_j is independent of condition C_i.

Construction of the dependency matrix. In order to determine whether an action A_j depends on a condition C_i, we have to compare the action values of A_j in those pairs of columns in the decision table for which the condition part only differs in the state for C_i. To this end, we introduce the concept of action value vector.

The *action value vector* of A_j with respect to state S_{ik} of C_i, AVV[i, k, j], is a vector which components are the values of A_j in those columns of the decision table in which the state of C_i is S_{ik}. The order of the components corresponds to the order of appearance of these values in the decision table, when it is scanned from the left to the right. For each $C_i \in$ **C** and for each $A_j \in$ **A**, there are n_i action value vectors AVV[i, k, j], k = 1 .. n_i. Because of the lexicographical order of the condition combinations in a decision table, the *lth* component of each AVV[i, k, j], k = 1 .. n_i, corresponds with the same combination of condition states for C_1, ..., C_{i-1}, C_{i+1}, ..., C_{cnum}. It follows that: an action A_j is independent of a condition C_i iff $\forall$ k $\in \{2, 3, ..., n_i\}$: AVV[i, k, j] = AVV[i, 1, j].

The algorithm to construct the dependency matrix of a decision table, is shown in figure 6.

```
for i := 1 to cnum do
 for j := 1 to anum do
  begin
   state := 1;
   equal := true;
   while (state < n_i) and equal do
    begin
     state := state + 1;
     equal := AVV[i, state, j] = AVV[i, 1, j];
    end;
   if equal then D[i, j] := 0 else D[i, j] := 1;
  end;
```

Figure 6: Algorithm to construct the dependency matrix of a decision table

Example. The dependency matrix D, corresponding with the decision table in figure 1 is:

	A_1	A_2	A_3
C_1	1	1	1
C_2	1	1	1
C_3	1	0	1
C_4	1	0	1

■

The algorithms that are presented in the remainder of this text assume that the dependency matrix contains no rows or columns with only '0' entries. Stated differently, it is supposed that *irrelevant conditions* (conditions on which none of the actions depend) and *independent actions* (actions that are independent of all conditions) are removed from the table before a decomposition algorithm is applied.

4.2. Decomposition in CA-disjoint decision tables

In order to determine whether a decision table DT can be decomposed into two or more CA-disjoint decision tables, the ROC2 algorithm of King and Nakornchai [3] can be applied on the dependency matrix of DT. King and Nakornchai have developed this algorithm in the context of group technology, that involves techniques to decompose large manufacturing systems into smaller systems. Their algorithm was also used by Moily and Murray [4] to modularize prolog knowledge bases.

The algorithm operates on a binary matrix, in our case the dependency matrix (the machine-part matrix in group technology), and rearranges the rows and columns in such a way that a block diagonal form is obtained if one exists. From this form, mutually independent sets of conditions and actions can easily be identified. The ROC2 algorithm is shown in figure 7.

```
(* row reordering *)
for j := anum to 1 do
 begin
  locate the rows in D[j] with '1' entries and note their relative order;
  move the located rows to the top of the matrix while maintaining their relative
  order;
 end;

(* column reordering *)
for i := cnum to 1 do
 begin
  locate the columns in D[i] with '1' entries and note their relative order;
  move the located columns to the left of the matrix while maintaining their relative
  order;
 end;
```

Figure 7: The ROC2 algorithm

Example. Consider the decision table in figure 8. Figure 9 shows the dependency matrix of this decision table, while figure 10 shows the row and column reordered matrix.

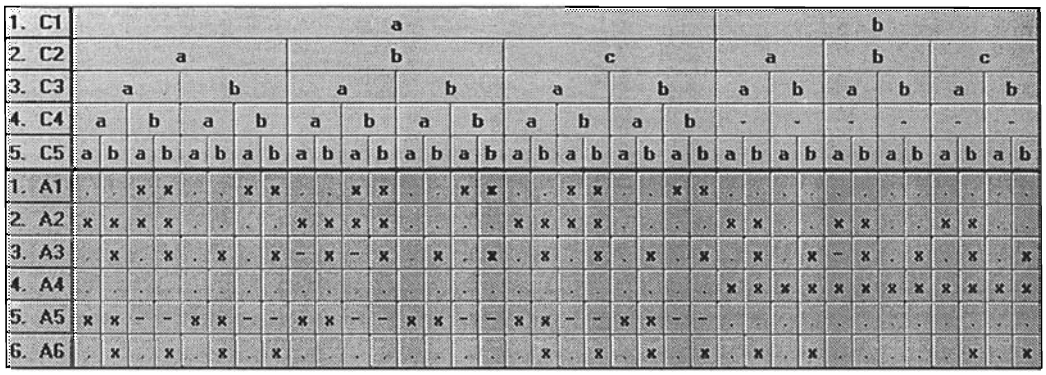


Figure 8

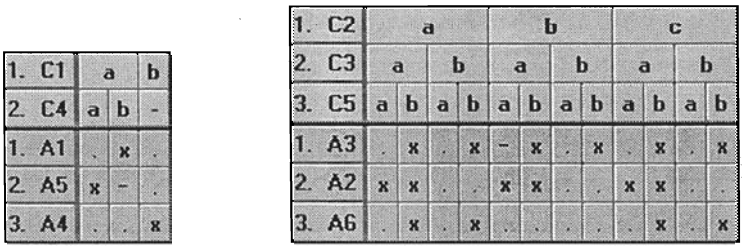
	A_1	A_2	A_3	A_4	A_5	A_6
C_1	1	0	0	1	1	0
C_2	0	0	1	0	0	1
C_3	0	1	1	0	0	0
C_4	1	0	0	0	1	0
C_5	0	0	1	0	0	1

Figure 9: Dependency matrix

	A_1	A_5	A_4	A_3	A_2	A_6
C_1	1	1	1	0	0	0
C_4	1	1	0	0	0	0
C_3	0	0	0	1	1	0
C_2	0	0	0	1	0	1
C_5	0	0	0	1	0	1

Figure 10: Reordered matrix

From figure 10, the following CA-disjoint decomposition is obvious:



■

4.3. Construction of action tables

The decomposition of a decision table DT into two decision tables that are connected by means of an action-head table structure is possible if there is a proper subset **A'⊂A** and a non-empty proper subset **C'⊂C** such that:

1. Actions of **A'** only depend on conditions of **C'**.
2. Consider for each condition combination of conditions in **C'** the subtable of DT determined by this condition combination and the actions in **A\A'**. All these subtables that are relevant (i.e. that contain for at least one action the value 'x' or '-') have to be equal.

If these requirements are met, DT can be decomposed into a head table, having **C'** as condition set and the actions of **A'** and a new introduced action that will serve as connecting action as action set. The condition set of the action table is **C\C'** and its action set is **A\A'**.

The algorithm to determine all subsets **C'** and **A'** that satisfy the above requirements is given in figure 11. It determines for each non-empty proper subset **C'⊂C**, on the basis of the dependency matrix, the action subset **A'** of actions that only depend on conditions of **C'**. Each in this way obtained condition subset **C'** and action subset **A'** satisfy requirement 1. Subsequently, it is checked whether they satisfy requirement 2.

```
for each non-empty proper subset C'⊂C do
 begin
  A' := ∅;
  for j := 1 to anum do
   if the subset of conditions that correspond with a '1' entry in D[j] is contained
      in C'
   then A' := A' ∪ {A_j};
  select the first condition combination of conditions in C';
  select the subtable of DT, determined by this condition combination and A\A';
  while the selected subtable is not relevant do
   begin
    select the next condition combination of conditions in C';
    select the subtable of DT, determined by this condition combination and A\A';
   end;
  checktable := subtable;
  check := true;
  while (not all condition combinations of conditions in C' are treated) and check do
   begin
    select the next condition combination of conditions in C';
    select the subtable of DT, determined by this condition combination and A\A';
    if the selected subtable is relevant
    then check := checktable = subtable;
   end;
  if check
  then constructing an action table on the basis of C' and A' is possible;
 end;
```

Figure 11: Algorithm to construct action tables

Example. Consider the following decision table and its dependency matrix:

1. C1	a												b											
2. C2	a				b				c				a				b				c			
3. C3	a		b		a		b		a		b		a		b		a		b		a		b	
4. C4	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b
1. A1	x	x	-	-	x	x	-	-	x	x	x	x	x	x	x	x	x	x	-	-	x	x	-	-
2. A2	x	x	x	x	-	-	-	-	x	x	x	x	-	-	-	-	x	x	-	-	-	-	-	-
3. A3	x	-	.	.	.	.	x	-	x	-	.	.	.	.	.	.	x	-	x	-	x	-	x	-
4. A4	x	-	.	.	.	.	x	-	x	-	.	.	.	.	.	.	x	-	x	-	x	-	x	-

	A_1	A_2	A_3	A_4
C_1	1	1	1	1
C_2	1	1	1	1
C_3	1	1	1	1
C_4	0	0	1	1

The following decomposition is possible:

1. C1	a						b				
2. C2	a		b		c		a		b		c	
3. C3	a	b	a	b	a	b	-	a	b	a	b
1. A1	x	-	x	-	x	x	x	x	-	x	-
2. A2	x	x	-	-	x	x	-	x	-	-	-
3. ^A	x	-	-	x	x	-	-	x	x	x	x

1. C4	a	b
1. A3	x	-
2. A4	x	-

■

4.4. Construction of condition tables

Each proper subset C'⊂C that contains at least two elements can serve as a basis for the construction of a condition table for a decision table DT. The condition combinations of the conditions in C' divide DT into subtables. If the number of distinct subtables is not 'too large', it might be useful to construct a condition table.

As upper bound for the number of distinct subtables, the following heuristic might be used: number of distinct subtables ≤ 1/2 × number of condition combinations of conditions in C'. This heuristic guarantees that

1. the number of columns in the head table does not exceed half of the number of columns in DT, and
2. the total number of columns in the head table and the condition table does not exceed the number of columns in DT.

If a satisfactory subset C' is found, DT can be decomposed into a condition table with C' as condition set and a head table which condition set contains the conditions of C\C' and a new introduced condition that serves as connecting condition. The number of states of this connecting condition is determined by the number of distinct subtables that are identified.

The algorithm to determine all satisfactory subsets C' on the basis of the above heuristic, is given in figure 12. It calculates for each proper subset C'⊂C that contains at least two elements the number of distinct subtables and it checks whether this number is smaller than the required upper bound.

```
for each proper subset C'⊂C with |C'| ≥ 2 do
 begin
  select the first condition combination of conditions in C';
  select the subtable of DT determined by this condition combination;
  subtblRow[1] := subtable;
  noSubtbls := 1;
  while (not all condition combinations of conditions in C' are treated) and
        (noSubtbls ≤ 1/2 * number of condition combinations of conditions in C') do
   begin
    select the next condition combination of conditions in C';
    select the subtable of DT determined by this condition combination;
    i := 1;
    subtblFnd := false;
    while (i ≤ noSbtbls) and (not subtblFnd) do
     begin
      subtblFnd := subtblRow[i] = subtable;
      i := i + 1;
     end;
    if not subtblFnd
    then
     begin
      noSubtbls := noSubtbls + 1;
      subtblRow[noSubtbls] := subtable;
     end;
   end;
  if noSubtbls ≤ 1/2 * (number of condition combinations of conditions in C')
  then constructing a condition table on the basis of C' is possible;
 end;
```

Figure 12: Algorithm to construct condition tables

Example. Consider the following decision table:

1. C1	a												b																			
2. C2	a								b								a								b							
3. C3	a				b				a				b				a				b				a				b			
4. C4	a		b		a		b		a		b		a		b		a		b		a		b		a		b		a		b	
5. C5	-	a	b	a	b	a	b	-	a	b	a	b	a	b	a	b	a	b	a	b	-	a	b	a	b	a	b	a	b	a	b	a	b
1. A1	x			-	-			x			x	x	x	-	x	x	x	-	x			x	x	x	-	-	-					
2. A2		x			x				x					-				-		x					-			x				
3. A3		-	-			x			-	-		x	x			x	x			-	-		x	x					x			
4. A4		-					x		-			x				x				-			x							x		

The following decomposition is possible:

1. C1	a								b																							
2. C2	a				b				a				b																			
3. C3	a		b		a		b		a		b		a		b																	
1. C := a	x				x						x																					
2. C := b			x												x																	
3. C := c							x		x				x																			

1. ^C	a								b																							
2. C4	a		b		a		b		a		b		a		b																	
3. C5	-	a	b	a	b	a	b	-	a	b	a	b	a	b	a	b																
1. A1	x			-	-			x			x	x	x	-																		
2. A2		x			x				x					-																		
3. A3		-	-			x			-	-		x	x																			
4. A4		-					x		-			x																				

■

4.5. Decomposition in C-connected decision tables

This algorithm starts from the dependency matrix and determines a partition of the action set such that the actions of each class depend on the same subset of conditions. Actions that depend on the same subset of conditions are found as equal columns in the dependency matrix. An overview of the algorithm is given in figure 13.

Once the partition is determined, the decision table can be decomposed in as much decision tables as there are classes in the partition. Under the supposition that the algorithm will only be applied to decision tables that cannot be decomposed in CA-disjoint decision tables, it follows that some of the component tables will be C-connected.

```
noClasses := 1;
class[1].column := D[1];
class[1].acts := {A1};
for j := 2 to anum do
 begin
  i := 0;
  classFnd := false;
  while (i < noClasses) and (not classFnd) do
   begin
    i := i + 1;
    classFnd := class[i].column = D[j];
   end;
```

(...)

(...)

```
if classFnd
 then class[i].acts := class[i].acts ∪ {A_j}
 else
  begin
   noClasses := noClasses + 1;
   class[noClasses].column := D[j];
   class[noClasses].acts := {A_j};
  end;
end;
```

Figure 13: Algorithm to decompose a decision table into C-connected decision tables

Example. Consider the following decision table and its dependency matrix:

1. C1	a				b				c			
2. C2	a		b		a		b		a		b	
3. C3	a	b	a	b	a	b	a	b	a	b	a	b
1. A1	.	.	.	.	x	x	.	.	x	x	.	.
2. A2	.	x	.	.	.	x	.	.	.	x	.	.
3. A3	.	.	x	x	.	.	.	.	.	.	-	-
4. A4	-	-	x	-	-	-	x	-	-	-	x	-

	A_1	A_2	A_3	A_4
C_1	1	0	1	0
C_2	1	1	1	1
C_3	0	1	0	1

The following decomposition is possible:

							C					
1. C1	a		b		c			1. C2	a		b	
2. C2	a	b	a	b	a	b		2. C3	a	b	a	b
1. A1	.	.	x	.	x	.		1. A2	.	x	.	.
2. A3	.	x	.	.	.	-		2. A4	-	-	x	-

■

5. Integration in the PROLOGA tool

A major drawback in the use of decision tables is the complexity of the manual construction process. Much redrawing work results from small changes like adding or deleting conditions, condition states and actions. Some manipulations, like the reordering of conditions, are quite impossible to perform manually. Therefore, a design tool for computer-supported construction, manipulation, validation and optimization of decision tables was built, called PROLOGA (PROcedural LOGic Analyzer).

Results with the PROLOGA tool show the possibility to acquire and verify knowledge in the form of a system of interrelated decision tables [5] [6]. Depending on the particularities of the application, this system of decision tables can be automatically implemented in several ways: conversion to a full knowledge based application with advanced consultation possibilities, program code generation and transformation into optimal test sequences, conversion to text and rules, etc. With this approach, the full trajectory of the development life cycle of an intelligent system is covered as complete as possible.

Constructing a decision table with the PROLOGA tool proceeds as follows. First, an empty decision table is built, based on the conditions, condition states and actions that can be identified in the problem situation. Then the decision table is filled. This can be done in two ways:

1. The knowledge engineer fills the decision table by marking for every decision column (combination of condition states) the appropriate action values.
2. The problem situation is translated into decision rules. Every decision rule, once defined, is immediately applied to the decision table.

Decision tables are automatically checked for empty columns, unreferenced conditions/actions, contradictions.

Decision tables that are related to the same problem situation can be grouped into one system. The process of structuring decision tables can be left to the knowledge engineer (manual decomposition) or it can be guided by the tool (automated decomposition). In the first case, the knowledge engineer decides on the structure and uses the tool's facilities to express the different relations between pairs of decision tables. In the second case, one large decision table can be built and the knowledge engineer can check whether it can be decomposed by one of the algorithms that were discussed in section 4. The system suggests possible decompositions which can then be accepted or rejected. Component tables can in turn be presented to the system for decomposition such that complex structures can be built. A combination of the manual and automated approach is also possible.

In order to enable a flexible construction and manipulation of decision tables, the following features are available throughout the modeling process: graphical display of the structure of decision tables, graphical display of the dependency matrix of a decision table, the possibility to navigate through the system by simply clicking the tables in the structure, etc.

Conclusion

It has been recognized in literature that the emerging problems of increasing complexity and maintenance of knowledge based systems can be tackled by the application of decomposition techniques. In this paper, it was demonstrated how the decomposition process of knowledge bases consisting of decision tables can be automated. Several algorithms were presented and their integration in the PROLOGA tool was discussed.

References

1. Colomb, R. M., Chung, Y. C. (1995) Strategies for Building Propositional Expert Systems, *International Journal of Intelligent Systems*, **10**, 295-328.
2. Hicks, R. C. (1995) Minimizing Maintenance Anomalies in Expert Systems, *Information & Management*, **28**, 177-184.
3. King, J. R., Nakornchai, V. (1982) Machine-Component Group Formation in Group Technology: Review and Extension, *International Journal on Production Research*, **20**(2), 117-133.
4. Moily, J. P., Murray, T. J. (1993) A Modularization Approach for Prolog Knowledge Bases, *Information Systems*, **18**(6), 405-417.
5. Vanthienen, J., Dries, E. (1994) Illustration of a Decision Table Tool for Specifying and Implementing Knowledge Based Systems, *International Journal on Artificial Intelligence Tools*, **3**(2), 267-288.
6. Vanthienen, J., Mues, C., Aerts, A., Wets, G. (1995) A Modularization Approach to the Verification of Knowledge Based Systems, *14th Int. Joint Conf. on Artificial Intelligence* (IJCAI95), *Workshop on Validation & Verification of Knowledge-Based Systems*, 96-102.
7. Wendler, B., Ayel, M. (1995) Verifying Dynamic Coherence in Modular Knowledge Bases, *European Symposium on the Validation and Verification of Knowledge-Based Systems* (EUROVAV95), 173-187.

Contents